

## Programming QST

QST is an apache module written in perl, and runs under modperl.

Modperl runs multithreaded which makes it easy to scale.

QST.pm has about 74,000 lines of code.

We use Textpad on windows to alter it, or Kate on linux.

If you make changes to QST.pm you must restart/start apache for the changes to take effect.

We do not use any requires or includes (other than a few perl modules we load at the beginning of QST.pm or in startup.pl), all the relevant code is in QST.pm.

We did it this way because of all the programs we had to reverse engineer over decades and we got tired of having to look for functions in all the requires and includes.

Apache passes the input string to QST.pm and we do our own parsing of it.

If you want to see the variables being passed to the webserver and thus QST.pm go to line 2157:

```
# REMOVE # BELOW FOR TROUBLE SHOOTING THEN UNCOMMENT the line further down
```

and do as instructed. Restart apache.

You read QST from the top down, and most of the file contains functions that are called from the Main Program which begins on about line 1665 and ends on about line 3096.

Below that are all the functions the main program calls along with database calls.

The top 1660 lines include the connection to the database, global variables, the %PASSED\_VALUES hash (all form elements must be in the hash along with a number for the size of input allowed), and the English language pack (begins line 163) and an additional language pack you can alter (ends line 1523).

All queries to QST are sent via POST in a form and the returned info is not cached.

- often the forms target a frame

QST uses frames to display pages.

In the forms there is always an **input name** and value (example: name=input value=make\_type).

The '**make\_type**' is the function called in QST to handle the request. The size of the input value cannot exceed 30 characters, all other names have different size values as seen in the %PASSED\_VALUES hash. This ensures there are not any buffer overruns.

The Main Program first checks if the info coming in is 'multipart/form-data' which is a file being uploaded or just normal input (begins approximately line 2155).

It then checks whether the user is logged in or not (begins about line 2440).

In all form requests there is a u\_id and session\_id. The u\_id is the number of the user when they were first created in the users table, and the session id is number they have been assigned as being logged in.

Their u\_id determines which role they have in QST. Each user may only have one role and each role has access to only functions appropriate to the role. An administrator only adds/changes/removes users/classes/courses/organizations, etc., a student can only take a QST, etc.. There can be no privilege escalation.

Practically all tables in the QST database have an auto increment row which is a primary index. Usually all queries are against this primary index, so we always are searching for a number and we make sure only numbers are used. This negates any sql injection.

Everything we search for in the database is based upon a number, with the uname field being the only exception (we limit this to alphanumeric values only) and it is a secondary index. All usernames must be unique. There are not any full table searches.

Once the user is logged in we get their role (approximately line 2611) and the classes they are attached to (approximately line 2737).

The appropriate function they are calling is listed under their role (approximately line 2771).

The Main Program ends on approximately line 3993.

Everything under the Main Program can only be called from the Main Program.

We use multilevel hashes (think records or objects) to hold information. This allows us to get a value by name, which is quite efficient.

QST is **basic** functional programming with html and style sheets written inline along with javascript. When a result is returned it is fully inclusive, there are no additional css files or javascript files required.

QST runs multithreaded and every now and then you may find something you write modperl may not like even if it is good code.

We write everything ourselves, that way it is easy to understand (as I mentioned, we hate to go looking for stuff or be expected to know something that is not included nor obvious – it's about the algorithm not the language). It ain't pretty, but it sure runs well.

Some functions will only do one thing and others may do many and call many other functions.

We could of reduced the size of QST by reusing the same functions in quizzes and tests and surveys, but it reached the point that surveys needed to be separate as there were just too many variables being passed to a single function in some cases.

We like to indent and you will find our opening and closing curly braces located differently.

If you are adding name/value pairs to forms, make sure to add them to the %PASSED\_VALUES hash along with the appropriate size to not overflow a field in the database if using it as an input value to a database field, nor consume much memory if only used as a variable. Also be sure to accept only verified information for that field, if it's a type integer remove anything that is not 0-9 before inputting. Decide what to include, not what to exclude.

QST is a very securely written program and we ask that you adhere to that principle if you decide to alter the code.

We would like to have a reputation based on security, easability, and scalability in the exam/assessment area.

If you have questions don't hesitate to contact us [qstsupport@shaw.ca](mailto:qstsupport@shaw.ca)

We will add to this document as people ask questions.

QSTonline.org